Midterm 3 - Exam C

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

Lecture section time:

Instructions:

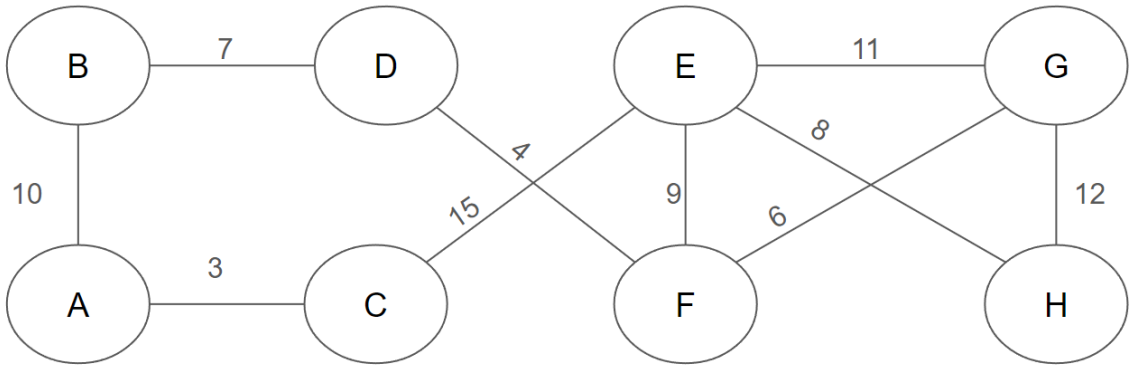
- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper. **Do not write your solutions in the back of pages!**
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing one extra page at the end if you need extra space. Make sure you mention in the space provided that you answer continues there.

Good luck!

Name:	PSU Access ID (xyz1234):

1 Kruskal's algorithm *(10 points)*

Use Kruskal's algorithm on the graph shown below. Ties can be broken arbitrarily.



(a) Edge (A, B) is added iteration:

○1 ○2 ○3 ○4 ○5 ○6 ●7 ○8 ○9 ○10 ○Never added

(b) Edge (A, C) is added iteration:

●1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○Never added

(c) Edge (B, D) is added iteration:

○1 ○2 ○3 ●4 ○5 ○6 ○7 ○8 ○9 ○10 ○Never added

(d) Edge (C, E) is added iteration:

○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ●Never added

(e) Edge (D, F) is added iteration:

○1 ●2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○Never added

(f) Edge (E, F) is added iteration:

○1 ○2 ○3 ○4 ○5 ●6 ○7 ○8 ○9 ○10 ○Never added

(g) Edge (E, G) is added iteration:

○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ●Never added

(h) Edge (E, H) is added iteration:

○1 ○2 ○3 ○4 ●5 ○6 ○7 ○8 ○9 ○10 ○Never added

Name:	**PSU Access ID (xyz1234):**

(i) Edge (F, G) is added iteration:

◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯Never added

(j) Edge (G, H) is added iteration:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ●Never added

Name:	**PSU Access ID (xyz1234):**

2 Horn Formula *(8 points)*

Find the variable assignment that solves the following horn formula if it exists. If there is no valid assignment, mark all as "No valid assignment". You must use the greedy algorithm from class to find the assignment.

$(w \wedge z) \Rightarrow x, (x \wedge z) \Rightarrow w, z \Rightarrow y, \Rightarrow z, (x \wedge y) \Rightarrow w, (\bar{w} \vee \bar{z}), (\bar{y})$

(a) w: ◯True ◯False ●No valid assignment

(b) x: ◯True ◯False ●No valid assignment

(c) y: ◯True ◯False ●No valid assignment

(d) z: ◯True ◯False ●No valid assignment

Name:	PSU Access ID (xyz1234):

3 Huffman Encoding *(8 points)*

Select the valid Huffman encoding for symbols a, b, c, d with the following multiplicities. You must choose that the right edges are 1 and the left edges are 0. While merging, the set with the larger number of unique symbols is the right child. If the sets are the same size, the set with the letter closest to the start of the alphabet is the right child.

$$a : 8, b : 6, c : 4, d : 3$$

(a) a: ●0 ◯1 ◯00 ◯01 ◯10 ◯11 ◯000 ◯001 ◯010 ◯011 ◯100 ◯101 ◯110 ◯111

(b) b: ◯0 ◯1 ◯00 ◯01 ●10 ◯11 ◯000 ◯001 ◯010 ◯011 ◯100 ◯101 ◯110 ◯111

(c) c: ◯0 ◯1 ◯00 ◯01 ◯10 ◯11 ◯000 ◯001 ◯010 ◯011 ◯100 ◯101 ◯110 ●111

(d) d: ◯0 ◯1 ◯00 ◯01 ◯10 ◯11 ◯000 ◯001 ◯010 ◯011 ◯100 ◯101 ●110 ◯111

Name:	PSU Access ID (xyz1234):

4 Edit distance *(10 points)*

Compute the edit distance between strings S_1 and S_2 by filling out the table below using the dynamic programming procedure. The edit distance is the number of insertions, deletions, and substitutions that are needed to turn one string into another. Note that points will not be awarded for filling out the table and it is only provided to help you answer. You must fill out the associated answer bubble for each square. Squares are numbered from left to right, top to bottom.

$$S_1 = GCCA, S_2 = AGC$$

	–	G	C	C	A
–	1.	2.	3.	4.	5.
A	6.	7.	8.	9.	10.
G	11.	12.	13.	14.	15.
C	16.	17.	18.	19.	20.

(a) Square 1 has which number in it?:

●0 ◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7

(b) Square 2 has which number in it?:

◯0 ●1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7

Name:	**PSU Access ID (xyz1234):**

(c) Square 3 has which number in it?:

◯0 ◯1 ●2 ◯3 ◯4 ◯5 ◯6 ◯7

(d) Square 4 has which number in it?:

◯0 ◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7

(e) Square 5 has which number in it?:

◯0 ◯1 ◯2 ◯3 ●4 ◯5 ◯6 ◯7

(f) Square 6 has which number in it?:

◯0 ●1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7

(g) Square 7 has which number in it?:

◯0 ●1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7

(h) Square 8 has which number in it?:

◯0 ◯1 ●2 ◯3 ◯4 ◯5 ◯6 ◯7

(i) Square 9 has which number in it?:

◯0 ◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7

(j) Square 10 has which number in it?:

◯0 ◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7

(k) Square 11 has which number in it?:

◯0 ◯1 ●2 ◯3 ◯4 ◯5 ◯6 ◯7

(l) Square 12 has which number in it?:

◯0 ●1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7

(m) Square 13 has which number in it?:

◯0 ◯1 ●2 ◯3 ◯4 ◯5 ◯6 ◯7

(n) Square 14 has which number in it?:

◯0 ◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7

(o) Square 15 has which number in it?:

◯0 ◯1 ◯2 ◯3 ●4 ◯5 ◯6 ◯7

(p) Square 16 has which number in it?:

◯0 ◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7

Name:	**PSU Access ID (xyz1234):**

(q) Square 17 has which number in it?:

◯0 ◯1 ●2 ◯3 ◯4 ◯5 ◯6 ◯7

(r) Square 18 has which number in it?:

◯0 ●1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7

(s) Square 19 has which number in it?:

◯0 ◯1 ●2 ◯3 ◯4 ◯5 ◯6 ◯7

(t) Square 20 has which number in it?:

◯0 ◯1 ◯2 ●3 ◯4 ◯5 ◯6 ◯7

Name:	PSU Access ID (xyz1234):

5 True/False *(10 points)*

True or false? Fill in the right bubble. No justification is needed.

T ● **F** ○ 1. Kruskal's algorithm can always create all possible MSTs of a graph across multiple runs.

T ○ **F** ● 2. Prim's algorithm is an approximate algorithm.

T ● **F** ○ 3. Running Kruskal's algorithm on a disconnected graph gives MSTs costing as much as the MSTs produced by running Kruskal's algorithm on each connected component in the graph.

T ● **F** ○ 4. A Huffman encoding is always prefix free.

T ○ **F** ● 5. If Huffman encoding gives a code of length n, there will never be anther symbol with a code length of $n+2$ unless there is also a symbol with a code length of $n+1$.

T ● **F** ○ 6. Huffman encoding can be implemented with a greedy algorithm.

T ○ **F** ● 7. The set cover problem can be solved optimally using a greedy algorithm.

T ○ **F** ● 8. The longest increasing subsequence problem can be solved in $O(n)$ with dynamic programming.

T ○ **F** ● 9. Edit distance of two strings of lengths M and N can be computed in $O(min\{M, N\})$ running time.

T ● **F** ○ 10. The longest common subsequence can be computed using dynamic programming.

Name:	PSU Access ID (xyz1234):

6 Longest decreasing subsequence *(15 points)*

Give an $O(n^2)$ running time algorithm to find the longest strictly decreasing subsequence of a sequence of n numbers. Explain your algorithm, the correctness, and its running time. Hint: Define dp[i] as the length of the longest decreasing subsequence that can be achieved using the first i numbers in the given array.

Solution: Given a list of numbers L, make a copy of L called L', and then sort L' in descending order. Then, just run the LCS algorithm (worksheet 11 question 4) on these two lists.

That is, define the following subproblems for all i, j such that $1 \leq i \leq n$, $1 \leq j \leq n$:

$$dp(i, j) = \text{length of longest common subsequence between } L[1, \cdots, i] \text{ and } L'[1, \cdots, j]$$

Then, the length of the LCS will be given by $dp(n, n)$. Initialize the dynamic program by setting $dp(i, 0) = 0$ and $dp(0, j) = 0$ for all i, j. The recursion to compute the values $dp(i, j)$ is the following:

$$dp(i, j) = \max\{dp(i-1, j), dp(i, j-1), dp(i-1, j-1) + \text{equal}(L_i, L'_j)\}$$

where equal(x, y) is 1 if x and y are the same character and is 0 otherwise.

The longest common subsequence must be decreasing because it is a subsequence of L' which is sorted. It is also the longest decreasing subsequence because being a subsequence of L' only adds the restriction that the subsequence must be decreasing. To make it strictly decreasing, remove duplicates of L' before running the LCS algorithm in linear time. Each time the LCS algorithm sets a $dp(i, j)$ the location it came from can be recorded $((i-1, j), (i, j-1), (i-1, j-1))$ so that backtracking can be done in linear time, noting that when $(i-1, j-1)$ is used, if $L[i] = L[j]$, then the element is added to the (reversed) LCS. The reversed LCS can then be reversed in linear time. Since $|L'| \leq |L| = n$, and sorting L can be done in $O(n^2)$ time, the final running time will be $O(|L||L'|) = O(n^2)$.

Alternate Solution: Variant of the DP algorithm for Longest increasing subsequence in lecture note suffices. The first condition for recurrence relation should be modified as $L(i) = 1 + max\{L(j) : 0 < j < i, a_j > a_i\}$. Other parts of the algorithm would be left the same.

Name:	PSU Access ID (xyz1234):

Spanning trees *(15 points)*

Assume you are given a graph $G = (V, E)$ with both positive and negative edge weights along with an algorithm A that can return a maximum spanning tree when given a graph with only *positive* edges. Describe an efficient way to transform G into a new graph G' containing only positive edge weights so that the maximum spanning tree of G can be easily found from the maximum spanning tree of G'. Prove that this procedure actually results in a maximum spanning tree and explain the running time.

Solution:

1. Find the smallest edge weight in the graph, denoted as w_l.
2. Update each edge weight W in the graph to a new weight $\hat{W}$, calculated as $\hat{W} = W - w_l + 1$.

As we subtract the smallest weight w_l from all edge weights and then add 1, the relative differences between the weights of the edges are maintained. Therefore this transformation ensures that all edge weights in G' are positive. As all the edges are positive in G', Algorithm A will return a valid MST of G'.

Proof: Consider any two edges e_1 and e_2 with original weights W_1 and W_2. Without loss of generality, assume $W_1 < W_2$. After the transformation, the weights become $\hat{W}_1 = W_1 - w_l + 1$ and $\hat{W}_2 = W_2 - w_l + 1$. We can see that $\hat{W}_1 < \hat{W}_2$ still holds, which means the ordering of the edges by weight is preserved. Consequently, algorithm A applied to G' will select the set of edges that would form a valid MST for G.

Running Time: The time complexity to find the minimum edge weight w_l is $O(|E|)$ where $|E|$ is the number of edges. Updating the weight of one edge takes $O(1)$ time. Updating all the edge weights takes $O(|E|)$ time. Hence, the total running time for the transformation is $O(|E|)$.

Name:	PSU Access ID:

Name:	PSU Access ID (xyz1234):